Backend Development

SWE 432, Fall 2017
Design and Implementation of Software for the Web
Real World Example

The hackers who broke into Equifax exploited a flaw in open-source server software

```java
1 /** The ContentTypeInfo Java class in Struts */
2 class ContentTypeHandler extends Interface {
3     ContentTypeHandler() {
4         this.hasQualifiedName("org.apache.struts2.rest.handler", "ContentTypeHandler")
5     }
6 }
7
8 /** The method `toObject` */
9 class ToObjectDeserializer extends Method {
10    ToObjectDeserializer() {
11        this.declaringType().getASupertype*( ) instanceof ContentTypeHandler and
12        this.getSignature = "toObject(java.io.Reader,java.lang.Object)"
13    }
14 }
```

A sample of code used by igtm to detect the vulnerability (igtm)

**Correction:** An earlier version of this article said the vulnerability exploited by the hackers who broke into Equifax was the one disclosed on Sep. 4. It's possible that the vulnerability that was targeted was one disclosed in March. We will update this post when we've confirmed which vulnerability it was.

The credit reporting agency Equifax announced on Sept. 7 that hackers stole records containing personal information on up to 143 million American consumers. The hackers behind the attack, the company said, “exploited a U.S. website application vulnerability to gain access to certain files.”

Today

• HW2 out, due next Tues before class

• Why do we need backends?
• Building backend web service with Node.js and Express
The "good" old days of backends

HTTP Request
GET /myApplicationEndpoint HTTP/1.1
Host: cs.gmu.edu
Accept: text/html

Runs a program

Web Server
Application

Here's some text to send back

My Application Backend

HTTP Response
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
History of Backend Development

• In the beginning, you wrote whatever you wanted using whatever language you wanted and whatever framework you wanted
• Then… PHP and ASP
  • Languages “designed” for writing backends
  • Encouraged spaghetti code
  • A lot of the web was built on this
• A whole lot of other languages were also springing up in the 90’s…
  • Ruby, Python, JSP
Backends today: Microservices

Browser
- Component presentation
- Component logic
- Front end framework
- HTTP Request
- HTTP Response (JSON)

Web Servers
- Microservice
- HTTP Request
- HTTP Response (JSON)

Database
Microservices

• Rather than horizontally scale identical web servers, vertically scale server infrastructure into many, small focused servers

• Some advantages

  • Fine-grained scalability: scale what services you need

  • Data-locality: data can be cached close to service providing functionality

  • Fault tolerance: restart only failing service rather than whole system

  • Reuse: use same micro service in multiple apps; use 3rd party rather than first party services
Why write a backend at all?
Why we need backends

• Security: SOME part of our code needs to be “trusted”
  • Validation, security, etc. that we don’t want to allow users to bypass

• Performance:
  • Avoid duplicating computation (do it once and cache)
  • Do heavy computation on more powerful machines
  • Do data-intensive computation “nearer” to the data

• Compatibility:
  • Can bring some dynamic behavior without requiring much JS support
Why Trust Matters

• Example: Transaction app

```javascript
function updateBalance(user, amountToAdd) {
    user.balance = user.balance + amountToAdd;
    fireRef.child(user.username).child("balance").set(user.balance);
}
```

• What’s wrong?
• How do you fix that?
Dynamic Web Apps

What the user interacts with:
- React
- HTML
- CSS
- JavaScript

What the front end interacts with:
- Firebase
- Some other API

Presentation:
- Some logic

“Back End”:
-Firebase
-Some other API

Data storage:
- Some other logic
Where do we put the logic?

Web "Front End"
- HTML
- CSS
- JavaScript
- React

"Back End"
- Firebase
- Some other API

Presentation

Some logic

Data storage

Some other logic

Frontend

Pros
- Very responsive (low latency)

Cons
- Security
- Performance
- Unable to share between front-ends

Backend

Pros
- Easy to refactor between multiple clients
- Logic is hidden from users (good for security, compatibility, and intensive computation)

Cons
- Interactions require a round-trip to server
HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web.

HTTP Request
- GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
- Host: cs.gmu.edu
- Accept: text/html

HTTP Response
- HTTP/1.1 200 OK
- Content-Type: text/html; charset=UTF-8
- <html><head>...

SWE 432 Section 002 Fall 2016 Syllabus and Schedule

"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:15pm Robinson Hall B228
Grades, Readings available as pdfs: Blackboard
Resources (Announcements, Schedule, Assignments, Discussion):
Piazza - https://piazza.com/gmu/fall2016/swe432001/home

Instructor: Prof. Jonathan Bell
bellj@gmu.edu
http://jonbell.net
Twitter: @jon_bell_
Office: 4422 Engineering Building; (703) 993-6089
Office Hours: Anytime electronically, Tues 10:30am-12:00pm, or by appointment
HTTP Requests

- Request may contain additional *header lines* specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

Other popular types:
- POST, PUT, DELETE, HEAD
Handling HTTP Requests in Express

- Node.js package for expressing rules about how to handle HTTP requests
Handling requests with Express

**HTTP GET Request**

```
GET /myResource/endpoint HTTP/1.1
Host: myHost.net
Accept: text/html
```

```
app.get("/myResource/endpoint", function(req, res){
  //Read stuff from req, then call res.send(myResponse)
});
```

**HTTP POST Request**

```
POST /myResource/endpoint HTTP/1.1
Host: myHost.net
Accept: text/html
```

```
app.post("/myResource/endpoint", function(req, res){
  //Read stuff from req, then call res.send(myResponse)
});
```
const express = require('express');
Import the module express

const app = express();
Create a new instance of express

const port = process.env.port || 3000;
Decide what port we want express to listen on

app.get('/', (req, res) => {
  var course = { name: 'SWE 432' };
  res.send(`Hello ${course.name}!`);
});

Create a callback for express to call when we have a “get” request to “/“. That callback has access to the request (req) and response (res).

app.listen(port, function () { });
Tell our new instance of express to listen on port.
Core concept: Routing

• The definition of end points (URIs) and how they respond to client requests.
  • **app.METHOD**(PATH, HANDLER)
  • METHOD: all, get, post, put, delete, [and others]
  • PATH: string
  • HANDLER: call back

```javascript
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```
Route paths

• Can specify strings, string patterns, and regular expressions
  • Can use ?, +, *, and ()
  • Matches request to root route

```javascript
app.get('/', function (req, res) {
  res.send('root');
});
```

• Matches request to /about

```javascript
app.get('/about', function (req, res) {
  res.send('about');
});
```

• Matches request to /abe and /abcde

```javascript
app.get('/ab(cd)?e', function (req, res) {
  res.send('ab(cd)?e');
});
```
Route parameters

- Named URL segments that capture values at specified location in URL
  - Stored into `req.params` object by name
- Example
  - Route path `/users/:userId/books/:bookId`
  - Request URL `http://localhost:3000/users/34/books/8989`
  - Resulting `req.params`: `{ "userId": "34", "bookId": "8989" }`

```javascript
app.get('/users/:userId/books/:bookId', function(req, res) {
  res.send(req.params);
});
```
Request object

- Enables reading properties of HTTP request
  - `req.body`: JSON submitted in request body *(must define body-parser to use)*
  - `req.ip`: IP of the address
  - `req.query`: URL query parameters
HTTP Responses

• Larger number of response codes (200 OK, 404 NOT FOUND)
• Message body only allowed with certain response status codes

“OK response”
Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

“HTML returned content”
Common MIME types:
application/json
application/pdf
image/png

[HTML data]
Response object

• Enables a response to client to be generated
  • `res.send()` - send string content
  • `res.download()` - prompts for a file download
  • `res.json()` - sends a response w/ application/json Content-Type header
  • `res.redirect()` - sends a redirect response
  • `res.sendStatus()` - sends only a status message
  • `res.sendFile()` - sends the file at the specified path

```javascript
app.get('/users/:userId/books/:bookId', function(req, res) {
  res.json({ "id": req.params.bookID });
});
```
Describing Responses

• What happens if something goes wrong while handling HTTP request?
  • How does client know what happened and what to try next?
• HTTP offers response status codes describing the nature of the response
  • 1xx Informational: Request received, continuing
  • 2xx Success: Request received, understood, accepted, processed
    • 200: OK
  • 3xx Redirection: Client must take additional action to complete request
    • 301: Moved Permanently
    • 307: Temporary Redirect

Describing Errors

• 4xx Client Error: client did not make a valid request to server. Examples:
  • 400 Bad request (e.g., malformed syntax)
  • 403 Forbidden: client lacks necessary permissions
  • 404 Not found
  • 405 Method Not Allowed: specified HTTP action not allowed for resource
  • 408 Request Timeout: server timed out waiting for a request
  • 410 Gone: Resource has been intentionally removed and will not return
  • 429 Too Many Requests
Describing Errors

• 5xx Server Error: The server failed to fulfill an apparently valid request.
  • 500 Internal Server Error: generic error message
  • 501 Not Implemented
  • 503 Service Unavailable: server is currently unavailable
Error handling in Express

• Express offers a default error handler

• Can specific error explicitly with status
  • `res.status(500);`
Making a request....

HTTP Request
GET /syllabus/syllabi-fall16/SWE432BellJ.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html

HTTP Response
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...

SWE 432 Section 002 Fall 2016 Syllabus and Schedule
"Design and Implementation of Software for the Web"

Class Hours: Tuesdays and Thursdays, 12:00pm-1:15pm  Robinson Hall B228
Grades, Readings available as pdfs: Blackboard
Resources (Announcements, Schedule, Assignments, Discussion):
Piazza - https://piazza.com/gmu/fall2016/swe432/home

Instructor: Prof. Jonathan Bell
bellj@gmu.edu
http://jonbell.net
Twitter: @jon_bell_
Office: 4422 Engineering Building; (703) 993-6089
Office Hours: Anytime electronically, Tues 10:30am-12:00pm, or by appointment
Making HTTP Requests w/ fetch

Install

```bash
npm install node-fetch --save
```

```javascript
var fetch = require('node-fetch');
fetch('https://api.github.com/users/github')
  .then(function(res) {
    return res.json();
  }).then(function(json) {
    console.log(json);
});
```

```javascript
var fetch = require('node-fetch');
fetch('https://github.com/');
  .then(function(res) {
    return res.text();
  }).then(function(body) {
    console.log(body);
});
```

https://www.npmjs.com/package/node-fetch
Demo: Example Express Microservice
Readings for next time

- Overview of HTTP: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

- Intro to REST: https://www.infoq.com/articles/rest-introduction